

## Appendix A: exemplary source code for thor32.dll.

```

//-----
#include <windows.h>
#include <TriceratMessaging.h>
HINSTANCE InstanceHandle;
bool Processed = false;
extern "C" __declspec(dllexport) void LoadThorA();
extern "C" __declspec(dllexport) void UnloadThorA();
// Shared Data
#pragma data_seg(".shared") // Make a new section that we'll make shared
HHOOK hHook = 0; // HHOOK from SetWindowsHook
#pragma data_seg()
LRESULT CALLBACK GetMsgProc(int code, WPARAM wParam, LPARAM lParam);
#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
    HWND hWndMjolnir = NULL;
    InstanceHandle = hinst;
    DisableThreadLibraryCalls(hinst);
    if (!Processed)
    {
        Processed = true;
        hWndMjolnir = FindWindow("TMjolnirMainForm", NULL);
        if (NULL != hWndMjolnir)
        {
            SendMessage(hWndMjolnir, TM_D2K_CHECKALLOWEDAPP, 0,
                GetCurrentProcessId());
        }
    }
    return true;
}
//-----
void LoadThorA()
{
    hHook = SetWindowsHookEx(WH_CALLWNDPROC, (HOOKPROC)GetMsgProc, InstanceHandle, 0);
}
//-----
void UnloadThorA()
{
    UnhookWindowsHookEx(hHook);
    hHook = NULL;
}
//-----
LRESULT CALLBACK GetMsgProc(int code, WPARAM wParam, LPARAM lParam)
{
    LRESULT retValue = 0;
    HWND hWndMjolnir = NULL;
    if (!Processed)
    {
        Processed = true;
        hWndMjolnir = FindWindow("TMjolnirMainForm", NULL);
        if (NULL != hWndMjolnir)
        {
            SendMessage(hWndMjolnir, TM_D2K_CHECKALLOWEDAPP, 0,
                GetCurrentProcessId());
        }
    }
    retValue = CallNextHookEx(hHook, code, wParam, lParam);
    return retValue;
}
//-----

```

## 5 Appendix B: exemplary source code for mjolnir.exe.

```

//-----
#include <vcl.h>
#pragma hdrstop
#include "MjolnirMainUnit.h"
10 #include "UnallowedAppUnit.h"
//-----
#pragma package(smart_init)
#pragma link "NetworkInfo"
#pragma link "StBase"
15 #pragma link "StShBase"
#pragma link "StTrIcon"
#pragma link "NetworkInfo"
#pragma resource "*.dfm"
#pragma link "psapi.lib"
20 typedef __stdcall bool (*LOADDLL)();
typedef __stdcall bool (*UNLOADDLL)();
static bool KillUserProcess(DWORD ProcessId);
TMjolnirMainForm *MjolnirMainForm;
//-----
25 __fastcall TMjolnirMainForm::TMjolnirMainForm(TComponent* Owner)
: TForm(Owner)
{
    PmpStarting = false;
    DesktopStarting = false;
30 SwingMjolnir = false;
}
//-----
void __fastcall TMjolnirMainForm::HookThor32()
{
35 if (NULL == hThor32Lib)
    {
        ShowMessage("Unable to load Thor32.Dll");
        Close();
    }
40 LOADDLL pfnLoadDll = (LOADDLL)GetProcAddress(hThor32Lib,
    "_LoadThorA");
    (*pfnLoadDll)();
}
void __fastcall TMjolnirMainForm::UnhookThor32()
45 {
    if (NULL == hThor32Lib)
    {
        ShowMessage("Unable to load Thor32.Dll");
        Close();
50 }
    UNLOADDLL pfnUnloadDll = (UNLOADDLL)GetProcAddress(hThor32Lib,
        "_UnloadThorA");
    (*pfnUnloadDll)();
}
55 void __fastcall TMjolnirMainForm::FormCreate(TObject *Sender)
{
    TRegistry *Reg = new TRegistry();
    bool ThorIsDisabled = false;
    ShowWindow(Application->Handle, SW_HIDE);
60 Session->Active = false;
    ProductID = TI_PRODUCT_DESK2K1;

```

```

5 Application->CreateForm(__classid(TLicenseForm), &LicenseForm);
  if (!LicenseForm->ValidateLicense())
  {
    MessageBox(NULL, "desktop 2001 License has expired!", "triCerat License",
      MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);
10 Application->Terminate();
    return;
  }
  delete LicenseForm;
  Reg->RootKey = HKKEY_LOCAL_MACHINE;
15 Reg->OpenKey("Software\\Tricerat\\Controls", true);
  try
  {
    ThorIsDisabled = Reg->ReadBool("DisableThor");
    if (ThorIsDisabled)
20 {
      Reg->CloseKey();
      delete Reg;
      Application->Terminate();
      return;
    }
  }
  catch(...)
  {
  }
30 Reg->CloseKey();
  Reg->OpenKey("Software\\Tricerat\\Desktop 2001", true);
  try
  {
    LoadTimer->Interval = Reg->ReadInteger("MjolnirStartupDelay") * 1000;
35 if (0 >= LoadTimer->Interval)
    {
      LoadTimer->Interval = 10000;
    }
  }
  catch(...)
40 {
    Reg->WriteInteger("MjolnirStartupDelay", 10);
    LoadTimer->Interval = 10000;
  }
  Reg->CloseKey();
  delete Reg;
  TSecurity *sec = new TSecurity();
  IsAdmin = sec->IsUserAdmin(getenv("COMPUTERNAME"), getenv("USERDOMAIN"),
  getenv("USERNAME"));
50 delete sec;
  hThor32Lib = LoadLibrary("Thor32.Dll");
  wts = new TWtsTools();
  HookThor32();
}
55 //-----
void __fastcall TMjolnirMainForm::FormActivate(TObject *Sender)
{
  ShowWindow(Application->Handle, SW_HIDE);
}
60 //-----
void __fastcall TMjolnirMainForm::FormClose(TObject *Sender, TCloseAction &Action)
{

```

```

5      UnhookThor32();
      if (NULL != hThor32Lib)
      {
          FreeLibrary(hThor32Lib);
      }
10     delete wts;
    }
    //-----
void __fastcall TMjolnirMainForm::FormHide(TObject *Sender)
{
15     ShowWindow(Application->Handle, SW_HIDE);
     Top = 5000;
     Left = 5000;
}
//-----
20 void __fastcall TMjolnirMainForm::HideTimerTimer(TObject *Sender)
{
     Hide();
}
//-----
25 void __fastcall TMjolnirMainForm::HookBitBtnClick(TObject *Sender)
{
     HookThor32();
}
//-----
30 void __fastcall TMjolnirMainForm::UnhookBitBtnClick(TObject *Sender)
{
     UnhookThor32();
}
//-----
35 void __fastcall TMjolnirMainForm::AddOwners(TStrings* sql)
{
     sql->Add(" IN (SELECT ID FROM Owners WHERE Name = " +
          FNetworkInfo->UserName + "");
     if (FNetworkInfo->LocalComputerName != ("\\\\\\" + FNetworkInfo->DomainName))
40     {
          FNetworkInfo->SourceServerName = FNetworkInfo->DomainControllerName;
          for (int i = 0; i < FNetworkInfo->MyGlobalGroupCount; i++)
              sql->Add(" OR Name = " + FNetworkInfo->MyGlobalGroupNames[i] + "");
     }
45     FNetworkInfo->SourceServerName = "";
     for (int i = 0; i < FNetworkInfo->MyLocalGroupCount; i++)
         sql->Add(" OR Name = " + FNetworkInfo->MyLocalGroupNames[i] + "");
     if (!FNetworkInfo->ClientName.IsEmpty())
         sql->Add(" OR Name = " + FNetworkInfo->ClientName + "");
50     if (!FNetworkInfo->LocalComputerName.IsEmpty())
         sql->Add(" OR Name = " + FNetworkInfo->LocalComputerName + "");
         sql->Add("");
}
//-----
55 void __fastcall TMjolnirMainForm::StringGridInitialize()
{
     AllowedAppsStringGrid->RowCount = 1;
     AllowedAppsStringGrid->FixedRows = 0;
     AllowedAppsStringGrid->ColCount = 3;
60     AllowedAppsStringGrid->ColWidths[0] = 100;
     AllowedAppsStringGrid->ColWidths[1] = 400;
     AllowedAppsStringGrid->ColWidths[2] = 50;
}

```

```

5     AllowedAppsStringGrid->Refresh();
    FirstRowOfStringGrid = true;
}
void __fastcall TMjolnirMainForm::LoadAllowedExecutables()
{
10     Session->Active = true;
    TQuery* query = new TQuery(NULL);
    AnsiString ParentProcess;
    AnsiString ProcessName;
    char szFileShortPath[MAX_PATH] = "unknown";
15     int InstanceLimit = 0;
    int j = 0;
    StringGridInitialize();
    query->UniDirectional = true;
    query->Constrained = true;
20     query->RequestLive = false;
    query->DatabaseName = "Tricerat D2K1";
    query->SQL->Add("SELECT DISTINCT e.Executable, e.InstanceLimit, e.Dependencies ");
    query->SQL->Add("FROM Executables e, StartMenuItems s, Owners o ");
    query->SQL->Add("WHERE ");
25     query->SQL->Add("e.ID = s.ExecutableID AND s.OwnerID = o.ID ");
    query->SQL->Add("AND e.Disabled = False ");
    query->SQL->Add("AND s.OwnerID ");
    AddOwners(query->SQL);
    try
30     {
        query->Open();
        for (int i = 0; i < query->RecordCount; i++)
        {
            try
35             {
                InstanceLimit = query->FieldByName("InstanceLimit")->AsInteger;
            }
            catch(...)
            {
40                 InstanceLimit = 1;
            }
            ProcessName = query->FieldByName("Executable")->AsString;
            if (0 != ExtractFileExt(ProcessName).AnsiCompareIC(".EXE"))
            {
45                 ProcessName = GetFileAssociation(ProcessName);
            }
            ParentProcess =
                ExtractFileName(ProcessName);
            AddAllowedApp(ParentProcess, ProcessName, InstanceLimit);
50             AddDependencies(query->FieldByName("Dependencies")->AsString,
                ParentProcess, 9999);
            query->Next();
        }
    }
55     catch (...)
    {
    }
    query->Close();
    query->SQL->Clear();
60     query->SQL->Add("SELECT DISTINCT e.Executable, e.InstanceLimit, e.Dependencies ");
    query->SQL->Add("FROM Executables e, DesktopItems d, Owners o ");
    query->SQL->Add("WHERE ");

```

```

5      query->SQL->Add("e.ID = d.ExecutableID AND d.OwnerID = o.ID ");
      query->SQL->Add("AND e.Disabled = False ");
      query->SQL->Add("AND d.OwnerID ");
      AddOwners(query->SQL);
      try
10     {
        query->Open();
        for (int i = 0; i < query->RecordCount; i++)
        {
            try
15            {
                InstanceLimit = query->FieldByName("InstanceLimit")->AsInteger;
            }
            catch(...)
            {
20                InstanceLimit = 1;
            }
            ProcessName = query->FieldByName("Executable")->AsString;

            if (0 != ExtractFileExt(ProcessName).AnsiCompareIC(".EXE"))
25            {
                ProcessName = GetFileAssociation(ProcessName);
            }
            ParentProcess =
                ExtractFileName(ProcessName);
            AddAllowedApp(ParentProcess, ProcessName, InstanceLimit);
            AddDependencies(query->FieldByName("Dependencies")->AsString,
                ParentProcess, 9999);
            query->Next();
        }
35     }
      catch (...)
      {
      }
      query->Close();
40     delete query;
      SwingMjolnir = true;
      Session->Active = false;
  }
  //-----
45  bool __fastcall TMjolnirMainForm::AddAllowedApp(AnsiString ParentProcess, AnsiString AppPath, int Instances)
  {
      //The TStringGrid has one row initialy, so don't add a new one.
      if (FirstRowOfStringGrid)
      {
50          FirstRowOfStringGrid = false;
      }
      else
      {
          AllowedAppsStringGrid->RowCount++;
55      }
      AllowedAppsStringGrid->Cells[0][AllowedAppsStringGrid->RowCount - 1] =
          ParentProcess;
      AllowedAppsStringGrid->Cells[1][AllowedAppsStringGrid->RowCount - 1] =
          ResolveFileShortPath(AppPath);
60      AllowedAppsStringGrid->Cells[2][AllowedAppsStringGrid->RowCount - 1] =
          String(Instances);
      return true;
  }

```

```

5      }
      void __fastcall TMjolnirMainForm::OnCheckAllowedApp(TMessage &Message)
      {
          ValidateProcess(Message.LParam);
      }
10     //-----
      bool __fastcall TMjolnirMainForm::ValidateProcess(DWORD ProcessId)
      {
          AnsiString ProcessPath;
          TStringList *RunningApps;
15         bool InstanceCountExceeded = false;
          bool ParentProcessRunning = false;
          bool ValidProcess = false;
          int i = 0;
          if (!SwingMjolnir)
20         {
              return true;
          }
          ProcessPath = GetProcessShortPath(ProcessId);
          if (ProcessPath.IsEmpty())
25         {
              return true;
          }
          //Get the list of running apps.
          RunningApps = wts->GetSessionProcessList();
          //Go through the Allowed Apps Grid and see if the Process is allowed to run.
          i = -1;
          while (AllowedAppsStringGrid->RowCount > ++i)
          {
30             if (0 == ProcessPath.AnsiCompareIC(
                AllowedAppsStringGrid->Cells[1][i]))
            {
                int AppCount = 0;
                int j = 0;
40                 //Check the instance count
                j = -1;
                while(RunningApps->Count > ++j)
                {
                    if (0 == RunningApps->Strings[j].AnsiCompareIC(
45                     ExtractFileName(AllowedAppsStringGrid->Cells[1][i]))
                    {
                        AppCount++;
                    }
                }
            }
50             if (AppCount > AllowedAppsStringGrid->Cells[2][i].ToIntDef(0))
            {
                InstanceCountExceeded = true;
            }
            //Try to find the Parent process.
55             j = -1;
            while(RunningApps->Count > ++j)
            {
                if (0 == RunningApps->Strings[j].AnsiCompareIC(
                    AllowedAppsStringGrid->Cells[0][i]))
60             {
                ParentProcessRunning = true;
            }
          }
      }

```

```

5      }
      }
      if (!InstanceCountExceeded && ParentProcessRunning)
      {
10         ValidProcess = true;
         break;
      }
    }
    RunningApps->Clear();
    delete RunningApps;
15    //Validate the Instance Count.
    if (InstanceCountExceeded)
    {
        if (IsAdmin)
        {
20            MessageBox(NULL, "The program Instance Count has been exceeded.\n\nPlease adjust the program \"Instance Count Limit\".",
                "Instance Count Exceeded", MB_OK | MB_SYSTEMMODAL | MB_ICONINFORMATION);
        }
        else
25        {
            if (KillUserProcess(ProcessId))
            {
                TInstanceLimitForm *Notify = new TInstanceLimitForm(NULL);
                Notify->ProcessPathLabel->Caption = ProcessPath;
                Notify->ShowModal();
                delete Notify;
            }
        }
        return false;
35    }
    //Validate the process.
    if (!ValidProcess)
    {
        if (IsAdmin)
40        {
            TAdminForm *admin = new TAdminForm(NULL);
            admin->ProcessEdit->Text = ProcessPath;
            admin->ShowModal();
            delete admin;
        }
45        else
        {
            if (KillUserProcess(ProcessId))
            {
50                TUnallowedAppForm *Notify = new TUnallowedAppForm(NULL);
                Notify->ProcessPathLabel->Caption = ProcessPath;
                Notify->ShowModal();
                delete Notify;
            }
55        }
        return false;
    }
    return true;
}
60 //-----
AnsiString __fastcall TMjolnirMainForm::ResolveFileShortPath(AnsiString File)
{

```



```

5   AnsiString Path;
   AnsiString FileShortPath;
   char szFileShortPath[MAX_PATH] = "unknown";
   TDirTools *Dir = new TDirTools();
   File = Dir->ParseEnvironment(File);
10  delete Dir;
   Path = getenv("PATH");
   Path = ".\\" + Path;
   if (ExtractFilePath(File).IsEmpty())
   {
15     //For some reason, FileSearch() does not search the CurrentDir.
     if (FileExists(GetCurrentDir() + "\\" + File))
     {
       File = GetCurrentDir() + "\\" + File;
     }
     else
20     {
       File = FileSearch(File, Path);
     }
   }
   GetShortPathName(File.c_str(), szFileShortPath,
   sizeof(szFileShortPath));
   FileShortPath = szFileShortPath;
   if (FileShortPath.IsEmpty())
   {
30     FileShortPath = File;
   }
   return FileShortPath;
}
//-----
35  AnsiString __fastcall TMjolinrMainForm::GetProcessShortPath(DWORD ProcessId)
   {
     HANDLE hProcess;
     HMODULE hMod;
     DWORD cbNeeded = 0;
40     char szProcessPath[MAX_PATH] = "unknown";
     char szProcessShortPath[MAX_PATH] = "unknown";
     AnsiString ProcessShortPath;
     hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE,
       ProcessId);
45     if (EnumProcessModules(hProcess, &hMod, sizeof(hMod), &cbNeeded))
     {
       //To get just the name of the process, call this:
       //GetModuleBaseName(hProcess, hMod, szProcessName, sizeof(szProcessName));
       //Get the full path of the process.
50       GetModuleFileNameEx(hProcess, hMod, szProcessPath, sizeof(szProcessPath));
       GetShortPathName(szProcessPath, szProcessShortPath,
         sizeof(szProcessShortPath));
       ProcessShortPath = szProcessShortPath;
     }
55     CloseHandle(hProcess);
     return ProcessShortPath;
   }
   //-----
60  bool KillUserProcess(DWORD ProcessId)
   {
     HANDLE hProcess;
     hProcess = OpenProcess(PROCESS_ALL_ACCESS, TRUE, ProcessId);

```

```

5      if (NULL == hProcess)
        return false;
        return TerminateProcess(hProcess, 0);
    }
    //-----
10 void __fastcall TMjolnirMainForm::OnDesktopInit(TMessage & Message)
    {
        if (0 == Message.WParam)
        {
            DesktopStarting = true;
            LoadTimer->Enabled = false;
15        }
        if (1 == Message.WParam)
        {
            DesktopStarting = false;
            if (!PmpStarting)
20            {
                InitializeMjolnir();
            }
        }
    }
25 void __fastcall TMjolnirMainForm::OnPmpInit(TMessage & Message)
    {
        if (0 == Message.WParam)
        {
            PmpStarting = true;
            LoadTimer->Enabled = false;
30        }
        if (1 == Message.WParam)
        {
            PmpStarting = false;
            if (!DesktopStarting)
35            {
                InitializeMjolnir();
            }
        }
40    }
    void __fastcall TMjolnirMainForm::InitializeMjolnir()
    {
        LoadAllowedExecutables();
45    }
    void __fastcall TMjolnirMainForm::OnRefresh(TMessage & Message)
    {
        InitializeMjolnir();
    }
50 void __fastcall TMjolnirMainForm::LoadTimerTimer(TObject *Sender)
    {
        LoadTimer->Enabled = false;
        InitializeMjolnir();
    }
55 //-----
    void __fastcall TMjolnirMainForm::RefreshBitBtnClick(TObject *Sender)
    {
        InitializeMjolnir();
    }
60 //-----
    void __fastcall TMjolnirMainForm::AddDependencies(AnsiString Delimited,
        AnsiString ParentProcess, int DefaultInstanceLimit)

```

```

5      {
        TStringList *ParsedStrings;
        int i = 0;
        if (Delimited.IsEmpty())
            return;
10     ParsedStrings = GetParsedStringList(Delimited);
        i = -1;
        while (ParsedStrings->Count > ++i)
        {
            AddAliasDependencies(ParsedStrings->Strings[i],
15             ParentProcess, DefaultInstanceLimit);
        }
    }
    TStringList * __fastcall TMjolnirMainForm::GetParsedStringList(AnsiString Delimited)
    {
20     TStringList *DelimitedCharList = new TStringList;
        TStringList *ParsedStrings = new TStringList();
        TStringList *SubStrings;
        AnsiString FoundString;
        AnsiString RemainingString;
25     bool SubStringsFound = false;
        int Index = 0;
        int i = 0;
        int j = 0;
        if (Delimited.IsEmpty())
30         return ParsedStrings;
        DelimitedCharList->Add(",");
        DelimitedCharList->Add(",");
        i = -1;
        while (DelimitedCharList->Count > ++i)
35     {
            Index = Delimited.AnsiPos(DelimitedCharList->Strings[i]);

            if (0 >= Index)
                continue;
40         SubStringsFound = true;
        FoundString = Delimited.SubString(1, Index - 1);
        RemainingString = Delimited.SubString(Index + 1,
            Delimited.Length() - Index);
        SubStrings = GetParsedStringList(FoundString);
45         j = -1;
        while (SubStrings->Count > ++j)
        {
            ParsedStrings->Add(SubStrings->Strings[j]);
        }
50         delete SubStrings;
        SubStrings = GetParsedStringList(RemainingString);
        j = -1;
        while (SubStrings->Count > ++j)
        {
55         ParsedStrings->Add(SubStrings->Strings[j]);
        }
        delete SubStrings;
    }
    if (!SubStringsFound)
60     {
        ParsedStrings->Add(Delimited);
    }

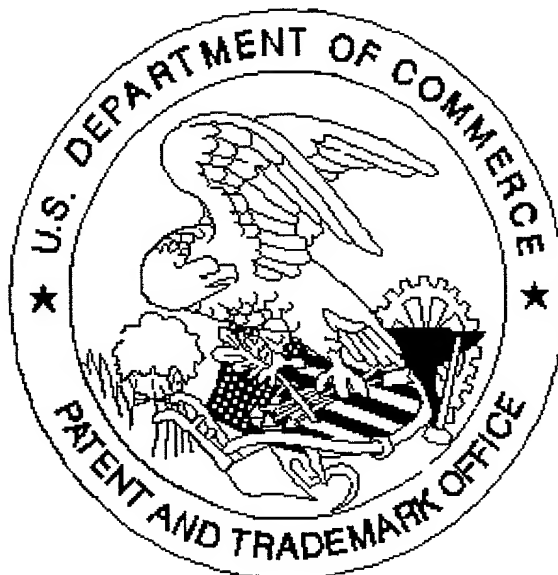
```

```

5      DelimitedCharList->Clear();
      delete DelimitedCharList;
      return ParsedStrings;
    }
void __fastcall TMjolnirMainForm::AddAliasDependencies(AnsiString Alias,
10      AnsiString ParentProcess, int DefaultInstanceLimit)
    {
        if (FileExists(ResolveFileShortPath(Alias)))
        {
            AddAllowedApp(ParentProcess, Alias, DefaultInstanceLimit);
            return;
15        }
        TQuery* query = new TQuery(NULL);
        query->UniDirectional = true;
        query->Constrained = true;
        query->RequestLive = false;
        query->DatabaseName = "Tricerat D2K1";
        query->SQL->Add("SELECT d.Path ");
        query->SQL->Add("FROM Dependencies d ");
        query->SQL->Add("WHERE ");
        query->SQL->Add("d.Name = '" + Alias + "'");
25      try
        {
            query->Open();
            for (int i = 0; i < query->RecordCount; i++)
            {
                AddAllowedApp(ParentProcess,
                    ResolveFileShortPath(query->FieldByName("Path")->AsString),
                    DefaultInstanceLimit);
                query->Next();
35            }
        }
        catch (...)
        {
        }
        query->Close();
        delete query;
40    }
AnsiString __fastcall TMjolnirMainForm::GetFileAssociation(AnsiString File)
    {
45      AnsiString Association;
      AnsiString FilePath;
      AnsiString FileName;
      char szResult[1024];
      ZeroMemory(szResult, sizeof(szResult));
      Association = File;
      File = ResolveFileShortPath(File);
      if (FileExists(File))
      {
          FileName = ExtractFileName(File);
          FilePath = ExtractFilePath(File);
          if (!FileName.IsEmpty() && !FilePath.IsEmpty())
          {
              if (32 < (int)FindExecutable(FileName.c_str(), FilePath.c_str(), szResult))
              {
                  if (FileExists(szResult))
60                  {
                      Association = szResult;
                      } } } } return Association;}

```

United States Patent & Trademark Office  
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☐ Page(s) \_\_\_\_\_ of \_\_\_\_\_ were not present  
for scanning. (Document title)

☐ Page(s) \_\_\_\_\_ of \_\_\_\_\_ were not present  
for scanning. (Document title)

☐ *Scanned copy is best available. FIG-1 IS DARK*